# Monte Carlo Path Tracing: Report

CS 775. Advanced Computer Graphics

**C Vishwesh**
140050031
vishwesh1696@gmail.com

**Suman Swaroop**
140050032
sumanswar12@gmail.com

February 17, 2018

## MULTIPLE SAMPLES PER PIXEL

### 1. XML files

Number of samples is read from the xml files. Under the image tag, set the "samples-per-pixel" int tag with the desired value. Samples per pixel value in xml file denotes the square root of the actual samples per pixel value.

### 2. Scene.cpp,

Introduced a new integer variable in image class, which needs to be initialized during object construction. In scene.cpp, during image object instantiating, parsed samples per pixel from xml file and passed that value to image object constructor.

### 2. Image.cpp

Divided each pixel into (samples-per-pixel) Now from each sub pixel , picked up a point uniformly sampled inside it, then we shoot rays from this point. Radiance returned for each such points is averaged out to give the resultant color to the pixel.

## AREA LIGHTS

### 1. Theory

Area light is circular disk uniquely identified by its center's position in world co-ordinate, length of its radius, normal direction and color.Any ray hitting the area light, returns the area light color without any further reflection and transmission because it is an emmissive surface. Area light has one more property of num samples that can be used to sample points on area light for shadow ray purposes.

### 2. Implementation

Area_light is a derived class of light_class.

**Area light::direct illumination** - We uniformly sampled n points on the area light and treated them like n point lights. The N point light's direct illumination are averaged out to get the resultant illumination from the area light. N is read from the xml file.
**AreaLight::sample**: Returns N uniformly sampled points on the disk.
**AreaLight::get_color:** return color of the area light as set in xml file.
Note we did not use shadow rays to compute soft shadows and neither used the direct illumination of the area light for the same because these were already accounted for by the monte carlo path tracing algorithm.

## MONTE CARLO PATH TRACING ALGORITHM

- **Step 1:** Shoot a ray from the camera

- **Step 2:** Find ray surface intersection
- **Step 3:** If no intersection return scene bg color.
- **Step 4:** if intersected object is a light source then return light color.
  else we sample the BRDF and BTDF to get the reflected and transmitted ray directions and return the radiance along these directions weighted with surface property constants.

## RADIANCE

### 1. Theory

We have considered three types of object.

- Perfectly reflecting object: When a ray falls on this object type, we calculate the reflected ray on the hit point, trace the radiance in the reflected ray direction and return the backtracked radiance. Set isreflect true in xml file to denote such objects.
- Perfectly Refracting objects: When a ray falls on this object type, we calculate the transmitted ray, check for TIR and accordingly trace the radiance. Set istransmit true in xml file to denote such objects.
- Normal objects: When a ray falls on this object type, BRDF and BTDF at the hit point are summed up, weighted with reflectance and transmittance coefficient set in xml files to get the indirect illumination at that point, illumination = transmittance * BTDF + reflectance * BRDF

### 2. Implementation

- BRDF: BRDF has two components diffuse and specular. For diffuse we sampled a direction in the hemisphere around the hit point by cosine weighted importance sampling then weighted the radiance of the sampled direction with kd of the surface. For specular, we sampled a direction from the conical lobe around the reflection direction and then weighted the radiance from the sampled direction with ks*(n+1)/(n+2).

  Now to decide whether to have specular or diffuse for this particular ray, we uniformly sampled a value between 0 and 1 and if the value sampled is less than kd/(kd+ks) then diffuse component is picked or else specular. kd , ks here are norm of color kd, ks read from xml file.
- BTDF: BTDF too has two components. Calculation of this is similar to BRDF except the hemisphere from which to sample direction changes. For diffuse we inverted the hemisphere normal around the surface. For specular the transmittance lobe is along the transmitted ray direction.
  Note: transmittance makes sense only when eta of object is defined properly.

## NEW PRIMITIVE : TORUS

Torus is uniquely defined by center position, axis direction and two radius components. Added a torus class inherited from object class with the above properties. Implemented torus Normal function and ray-torus intersection functions.

## NEW PRIMITIVE : TRIANGLE

Triangle is uniquely defined by the position of its three vertices. These are read from xml files. Added a triangle subclass inherited from object class with the above properties. Normal calculation and intersection with ray functions for this primitive is implemented.

## SCENE DESCRIPTION

### 1. Cornell Scene with Sphere primitives

This is the original cornell scene as in smallpt website.It has 6 big spheres to form the walls of box. One completely reflecting mirror sphere on left and a glass sphere on right.
Effects:

- Caustic effects: Glossy effect at the bottom of refracting sphere
- Bleeding effects: The back plane has some amount of red color on left side and some amount of blue on right side.
- Soft shadows: Shadow of spheres are soft.

### 2. Cornell Scene with Torus and pyramid

This scene also has 5 walls but uses large triangle primitive to render it. Inside this box, there is one torus primitives, and one triangular pyramid. Torus has a refracting surface. Pyramid has a high intense yellow colured surface.

### 3. Cornell Scene with Torus and triangle primitive

This scene also has 5 walls but uses large triangle primitive to render it. Inside this box, there are two torus primitives. The green torus primitive shows glossy specular effect and the other one is completely reflecting mirror.
Effects:Bleeding effects: On the back walls. Specular effect: Green Torus has a glossy specular effect on it.

## TIME

- The given scene files take around 8-10 mins to render on an i7 cpu.

## RESULTS

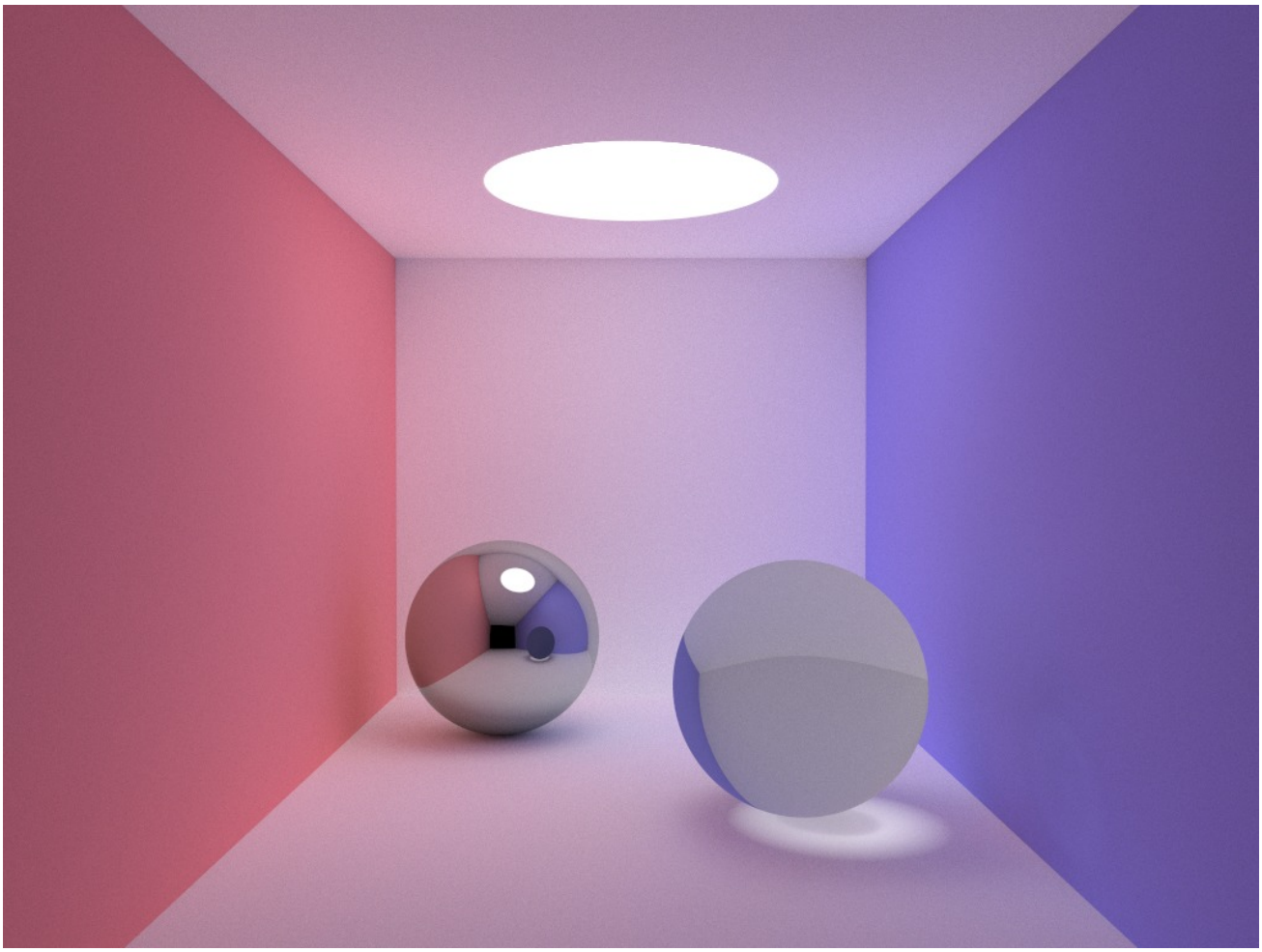

Figure 1: Cornell Scene 19600 samples per pixel

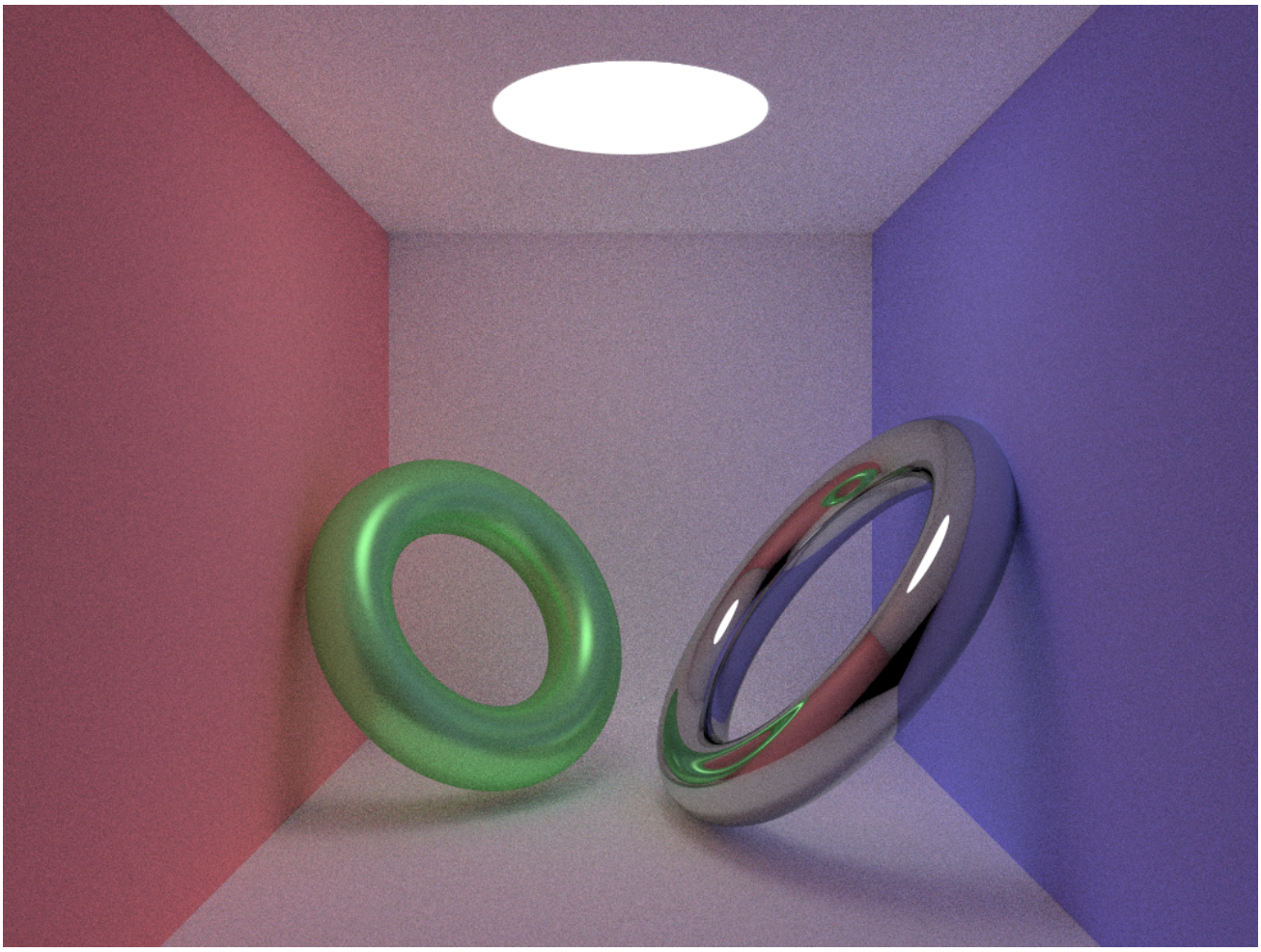

Figure 2: Scene1 2500 samples per pixel

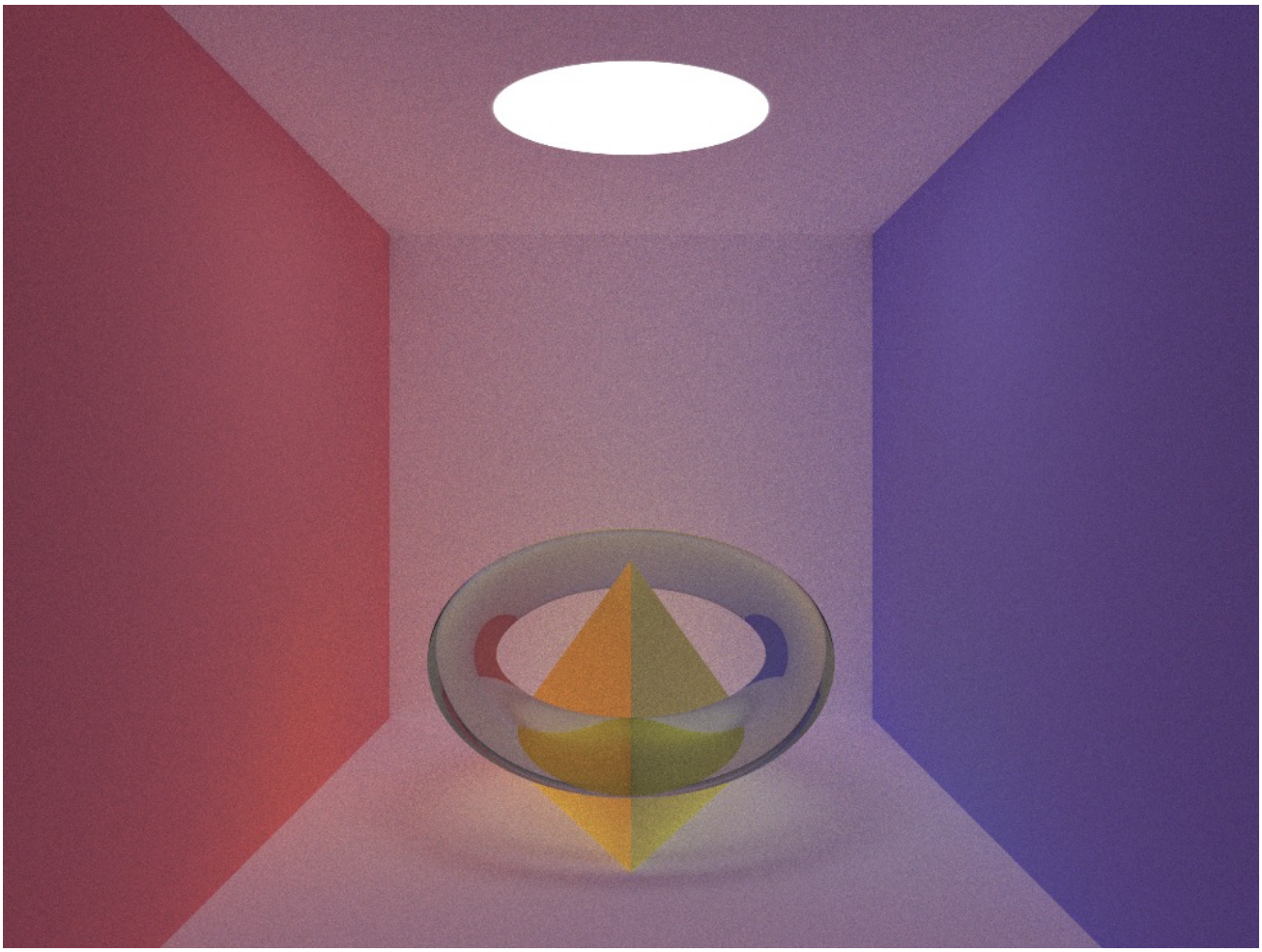

Figure 3: Scene2 2500 samples per pixel

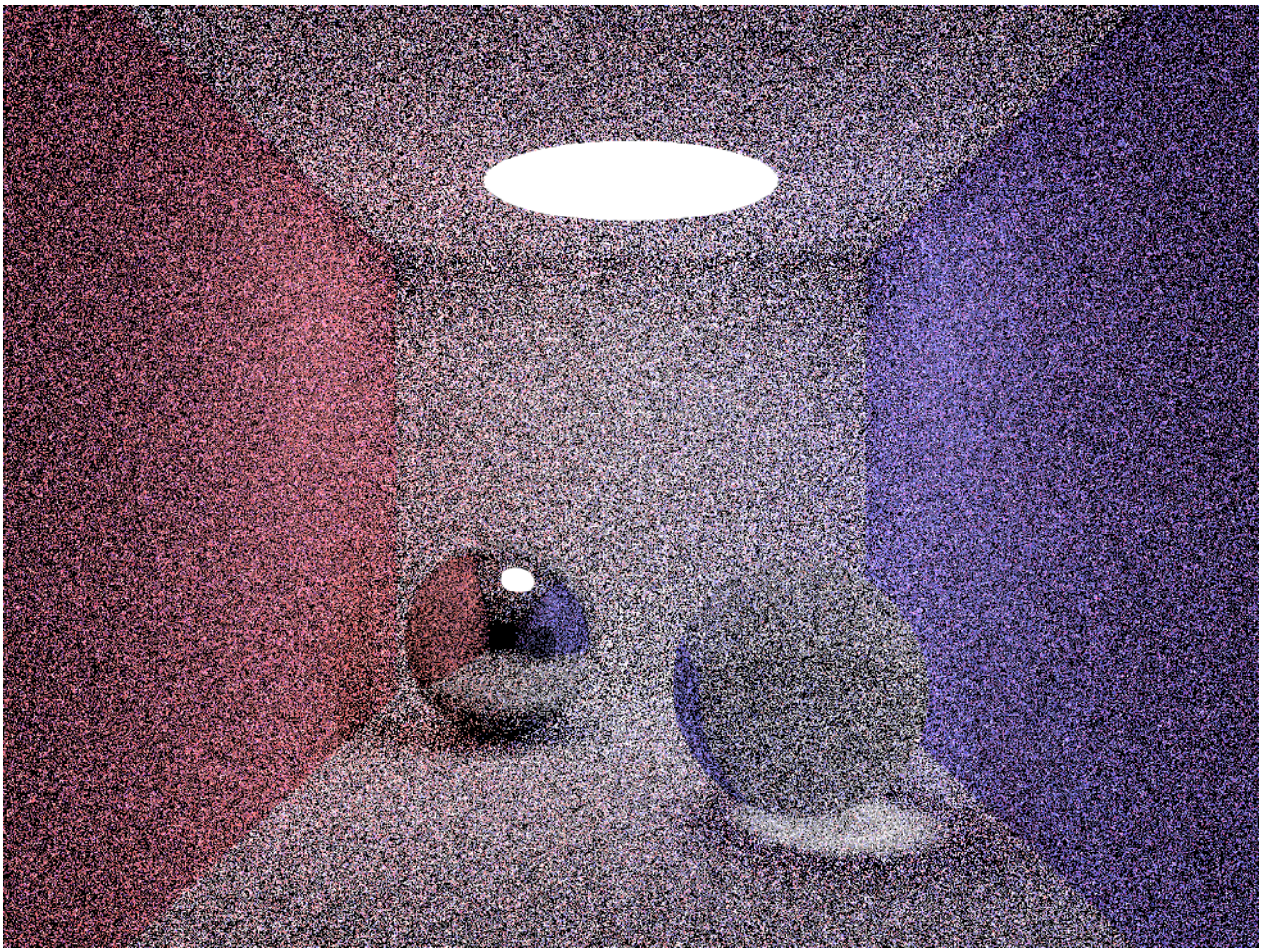

Figure 4: Cornell Scene 16 samples per pixel

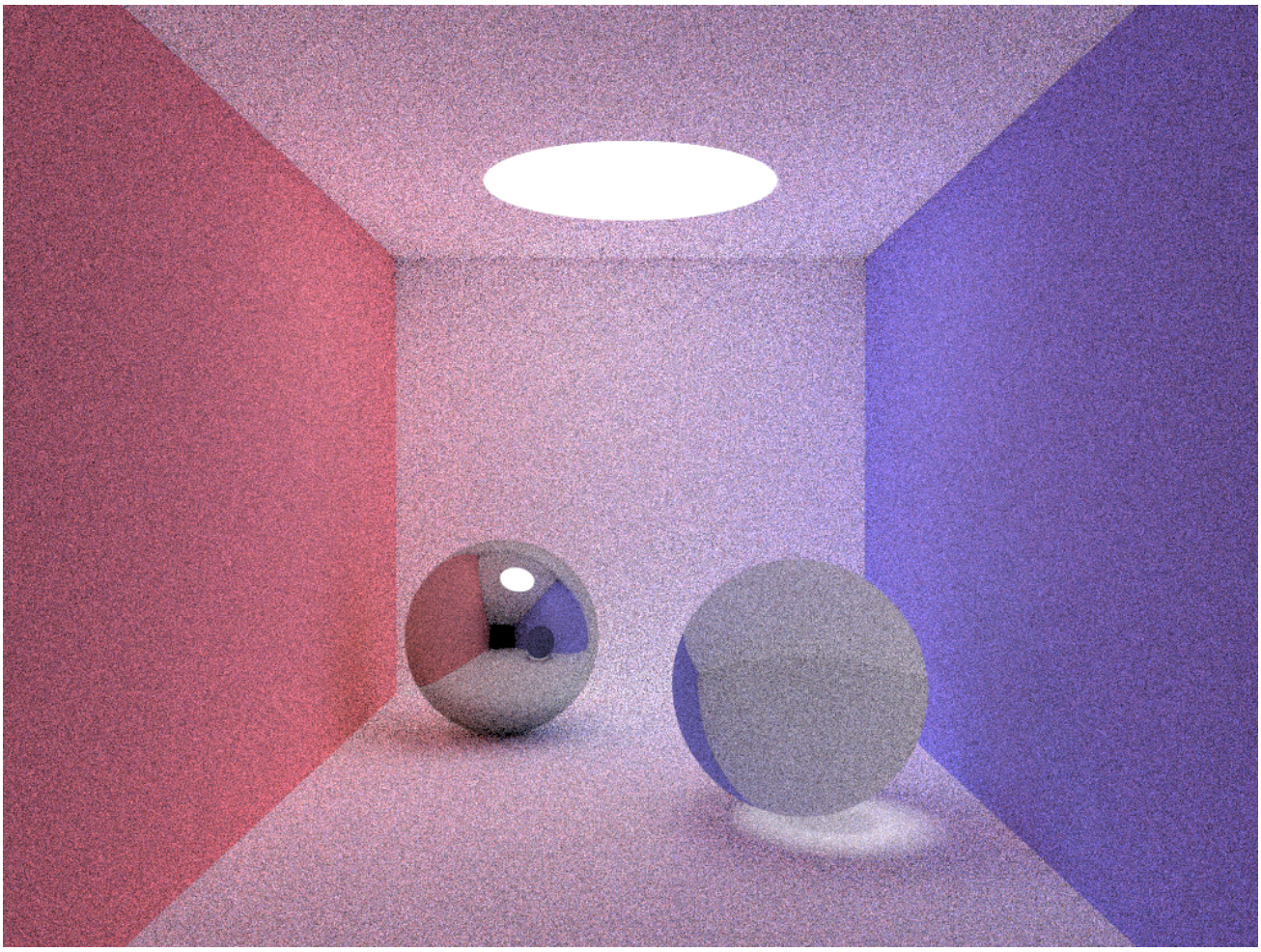

Figure 5: Cornell Scene 256 samples per pixel

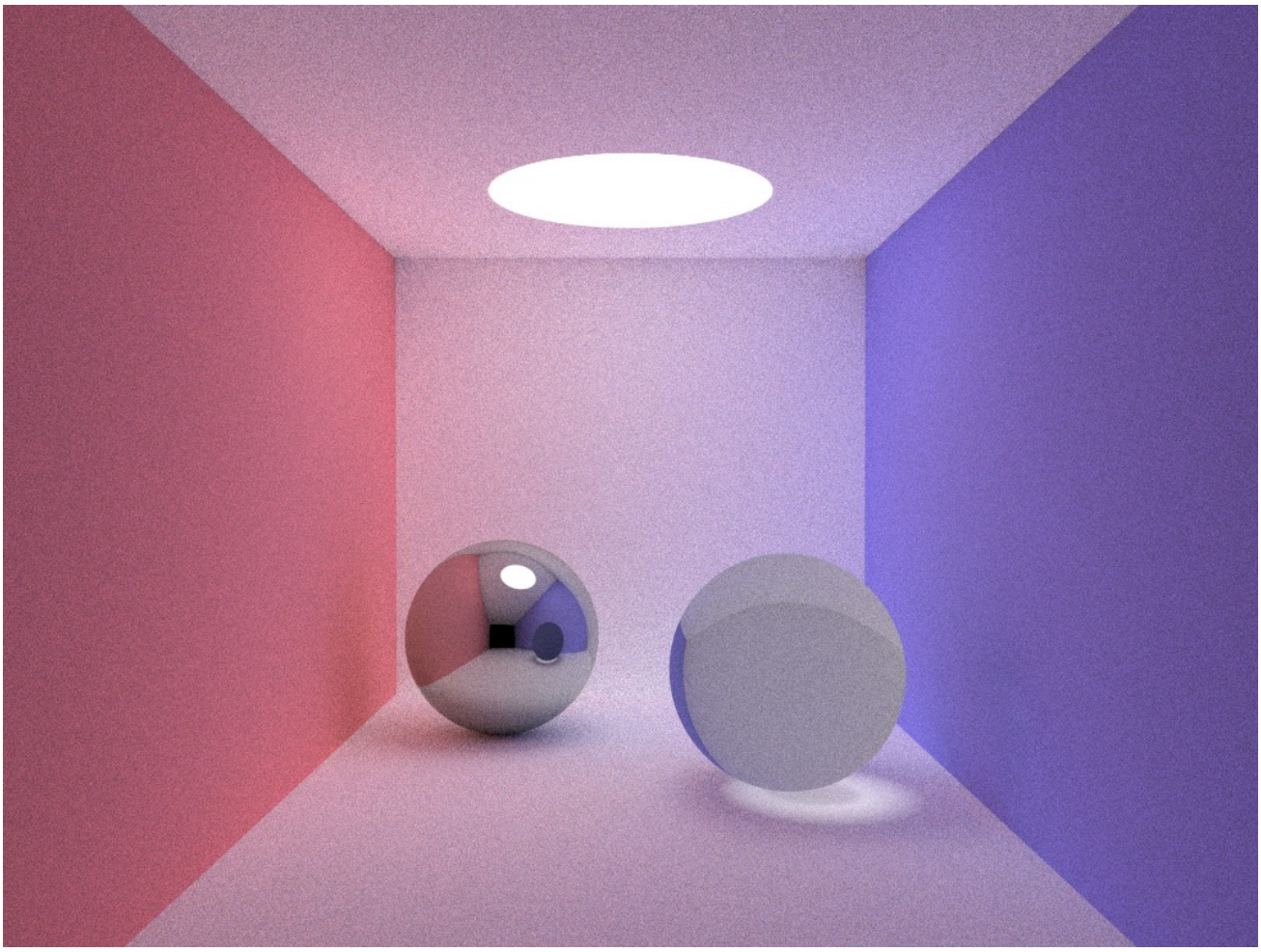

Figure 6: Cornell Scene 1600 samples per pixel